

manuel d'installation, d'aide à la
programmation et d'utilisation du logiciel
prplast2

Maxime Louvel, stagiaire sous la responsabilité de Mr. Laurent Baillet

4 août 2006

Table des matières

1	Installation	4
1.1	Linux	4
1.1.1	installation	4
1.1.2	compilation du logiciel prplast2	9
1.2	Autres systèmes d'exploitation	9
2	Utilisation	10
2.1	menu	10
2.2	Zoom	10
2.3	en général	11
3	Aide à la programmation	12
3.1	Makefile	12
3.2	philosophie de programmation	15
3.2.1	callbacks	15
3.3	Création d'une boîte	16
3.3.1	changements effectués depuis la version GKS	16
3.3.2	INCLUDE	17
3.3.3	organisation	18
3.3.4	initialisation	18
3.3.5	callbacks	23
3.3.6	remplissage de la boîte	24
3.3.7	fin, retour au menu supérieur	25
3.4	Explication des points récurrents	26
3.4.1	cadre	26
3.4.2	couleurs	27
3.4.3	dessin	27
3.4.4	écriture de texte	32
3.4.5	menu	32
3.4.6	souris	33
3.4.7	zoom	35

<i>TABLE DES MATIÈRES</i>	2
---------------------------	---

3.5 conseils, mise en garde	36
3.5.1 variables	36
3.5.2 fenêtre courante	37
3.5.3 rafraîchissement	37
A complément sur l'installation	39
A.1 glut3.7	39
A.2 f90gl	39
A.2.1 INSTALL	39
A.2.2 mf_key	45
B fichier piece.f90	47
C fichier despiece.f90	55
D corps des méthodes de zoom	56
D.1 zoom	56
D.2 écriture	60

Table des figures

3.1	tableau de correspondance valeur -> couleur	27
3.2	tableau de correspondance GKS -> openGL	31

Chapitre 1

Installation

1.1 Linux

1.1.1 installation

Ce document contient une description simplifiée des étapes à suivre pour installer la librairie **f90gl** permettant de faire de l'openGL avec le langage fortran 90. Il n'est ni complet, ni parfait, ni exhaustif. Si vous désirez plus d'informations, reportez vous à la documentation en ligne :
<http://math.nist.gov/f90gl/documentation.html>

Vous pouvez télécharger les « tar.gz » sur le site de la librairie :
<http://math.nist.gov/f90gl/software.html>

requit

Sur votre machine, doivent être installés les paquets de développement des librairies suivantes :

- X11
- MesaGl (ou autre distribution que Mesa)
- MesaGlu (ou autre que Mesa)
- **Attention** il ne faut pas installer de librairie Glut, ceci provoque des conflit avec celle que vous allez installer.

Vous devez aussi posséder un compilateur fortran (g95 de la GNU, ifort d'Intel ...) et un compilateur C (gcc de la GNU, ...).

Glut3.7

Téléchargez la version de Glut3.7 disponible sur le site de la librairie **f90gl** :

<http://math.nist.gov/f90gl/software.html>.

Décompressez ce fichier dans répertoire de travail.

```
...>cd /home/mlouvel && tar xzf glut-3.7.1.tar.gz
```

Rentrez ensuite dans le répertoire qui vient d'être créé.

```
...>cd glut-3.7.1
```

Nous utiliserons ici le script « **imakes** », pour utiliser le script « **sgi** » reportez vous à la documentation associée.

```
...>./mkmkfiles.imake
```

(Ne pas s'inquiéter s'il y a une erreur disant que le répertoire **./progs** n'existe pas car il n'y en a pas ...)

Ensuite compilez la librairie Glut3.7.

```
...>make
```

(Idem si la même erreur survient pas d'inquiétude)

Pour vérifier que la librairie a bien été compilée, vous pouvez vérifier que répertoire le **lib/glut** contient bien le fichier « **libglut.a** ».

```
...>ls lib/glut
```

ou encore

```
...>ls lib/glut | grep libglut.a
```

Si vous avez bien ce fichier alors vous pouvez passer à l'étape suivante : compilation des librairies de **f90gl**.

f90gl

Téléchargez la version de f90gl disponible sur le site de la librairie **f90gl** : <http://math.nist.gov/f90gl/software.html>. Décompressez ce fichier dans votre répertoire de travail.

```
...>cd /home/mlouvel && tar xzf f90gl-1.2.11.tar.gz
```

Rentrez ensuite dans le répertoire.

```
...>cd f90gl-1.2.11
```

Dans le fichier **INSTALL** se trouve les instructions pour installer la librairie **f90gl**, nous donnons ici une version simplifiée et surtout moins complète. (une copie du fichier **INSTALL** est disponible en dans l'annexe A.2.1, page 39).

Il convient maintenant de trouver le code de votre système, pour cela regardez dans le fichier « **mf_key** ». (une copie est disponible en Annexe A.2.2).

Ici nous travaillons sur un PC x86, avec Linux, un compilateur intel (ifort) et la distribution Mesa d'openGL. Notre code système est donc **8lim7**. Car nous avons un noyau <2.16, 6 s'il s'agit d'un noyau >= 2.16). Pour connaître la version de son noyau, exécutez la commande :
...>kernelversion

Une fois le code du système trouvé, il faut remplir le fichier **fppxxxx** qui se trouve dans le répertoire **/home/mlouvel/fppincs**. (xxxx est le code système précédemment mentionné)

Dans notre cas il s'agit du fichier **fpp8lim**.

Dans ce fichier, renseignez la version que vous utilisez d'openGL sous la forme : **#define OPENGL_v_v**, nous utilisons la version 4.0, nous mettons donc **#define OPENGL_4_0**.

Il faut maintenant modifier le fichier makefile correspondant à votre installation (**mfxxxx**), dans notre exemple : **mf8lim7**. Pour ceci modifiez ce qui est nécessaire dans la partie « User configuration parameters ».

mf8lim7 avant modification

```
#----- User configuration parameters ----- # the include directory(ies)
for OpenGL (GL is a subdirectory of it) and GLUT OGLINC = -I$(MESAHOME)/include
# the include directory for X11 X11INC = -I/usr/X11R6/include
# the library directory(ies) for OpenGL and GLUT OGLLIBDIR = -
L$(MESAHOME)/lib
# the OpenGL, GLU and GLUT libraries OGLLIB = -lglut -lGLU -lGL
```

```

# the library directory for X11
X11LIBDIR = -L/usr/X11R6/lib

# the X11 libraries
X11LIB = -lXaw -lXt -lXmu -lXi -lXext -lX11 -lm

# extra characters to put into the library names, in case there is # more
than one OpenGL installed. For example if LIBNAME is Mesa # then the
libraries will be libMesaGL.a etc. LIBNAME =

# commands for making a library MAKELIB = ar rcv
RANLIB = ranlib

# fortran 90 compiler and compiler flags
F90 = ifort
F90FLAGS = -O

# C compiler and compiler flags
CC = gcc
CFLAGS = -O -DFNAME=$(FNAME)

# The suffix for module files generated by the compiler. If none are #
generated, then use 'junk' or something (do not leave it blank).
MOD = mod

# The compiler flag to get at the module file for opengl_kinds in ../include/GL
USEMOD = -I../include/GL
#————— end of user configuration parameters —————

```

Liste des différents points à changer :

1. OGLINC : il faut indiquer au compilateur où se trouve les fichiers d'en-tête, les headers de la librairie glut.
Remplacez \$(MESAHOME)/include par **repTravail/glut-3.7.1/include**.
2. OGLLIBDIR : les librairies gl et glu sont installées dans
...>/usr/lib
, il n'y donc rien à faire normalement. Au contraire, pour la librairie glut il faut indiquer au compilateur où elle se trouve : dans votre répertoire de travail, **repTravail/glut-3.7.1/lib/glut** dans notre exemple :
...>/home/mlouvel/glut-3.7.1/lib/glut

Remplacez $\$(\text{MESAHOME})/\text{lib}$ par **repTravail/glut-3.7.1/lib/glut**.

3. X11INC : sous Linux, les librairies de X devraient se trouver dans le répertoire, donc ne changez rien. Si ces librairies se trouvent dans un autre répertoire, indiquez ce répertoire (contenant le headers).
4. X11LIBDIR : idem X11INC
5. CC : renseignez ici le nom de votre compilateur C ici, celui de la GNU est utilisé.
6. USEMOD : cette variable permet d'indiquer au compilateur où trouver les modules OpenGL.
Remplacez `../include/GL` par **repTravail/f90gl-1.2.11/include/**. Dans notre exemple : **/home/mlouvel/f90gl-1.2.11/include/**.

Note : les option -L et -I permettent de spécifier au compilateur où se trouve les librairies (resp. les headers) sont utilisées par les compilateur fortran **ifort** et **g95** mais je n'ai pas testé pour les autres compilateurs. Ainsi, si vous utilisez un autre compilateur, vérifiez que les options -L et -I sont bien implantées, sinon trouvez un équivalent.

Vous pouvez enfin compiler la librairie. Placez vous dans le répertoire de travail et taper la commande

```
...>make -f mfxxxx
```

où

```
...>xxxx
```

représente le code système. Dans notre exemple, nous tapons la commande

```
...>make -f mf8lim7
```

Si vous avez une erreur vous indiquant qu'une librairie X11, n'a pas put être "lié" (il m'est arrivé d'avoir ce problème, pour lXaw), cela vient probablement d'un problème d'installation. Vous pouvez essayer d'enlever cette librairie, qui n'est pas indispensable car dans le logiciel prplast2 nous n'utilisons pas toutes les fonctionnalités d'OpenGL.
Erreur renvoyée par le compilateur : « /usr/bin/ld : ne peut trouver -lXaw »
Cependant faites attention. Il est préférable de vérifier l'installation des librairies.

Si vous êtes arrivé ici, les librairies de **f90gl** devraient avoir compilées (avec quelques warning sûrement mais ...). Donc dernière étape de l'installation,

vérifiez que les librairies aient bien été compilées : dans le répertoire **repTravail/f90gl-1.2.11/lib** vous devriez avoir les trois fichiers suivant : **libf90GL.a libf90GLU.a libf90glut.a**. Si ceux-ci sont présents vous pouvez passer à la compilation du logiciel.

1.1.2 compilation du logiciel prplast2

Nous vous proposons un Makefile permettant de compiler tout le logiciel en une seule commande. Il nécessite d'être légèrement paramétré. Référez-vous à la section 3.1, page 12 pour savoir comment bien utiliser et mettre à jour ce Makefile.

En fait le paramétrage est plus ou moins identique à celui du Makefile permettant de compiler les librairies **f90gl**.

Si vous avez installé glut3.7 et f90gl-1.2.11 dans votre répertoire de travail alors modifiez uniquement la variable **REP_TRAVAIL**. Remplacez **/home/mlouvel** par votre répertoire de travail.

Il ne vous reste plus qu'à renseigner le nom du compilateur que vous utilisez en modifiant la variable **F90** : Remplacez « ifort » par votre compilateur. Vous pouvez aussi modifier les flags de compilation, par exemple, pour optimiser le code produit. Pour cela modifiez la variable **F90FLAGS**.

1.2 Autres systèmes d'exploitation

Faute de temps le programme n'a pu être testé sous d'autres systèmes, windows, sun, Hp. Le code openGL est portable, de même que le fortran, de plus la librairie **f90gl** est dite portable ; ainsi le passage sur un autre système ne devrait pas poser de problème. Pour installer les librairies (glut3.7, f90gl-1.2.11) référez-vous directement au site de la librairie **f90gl** : <http://math.nist.gov/f90gl/>

Chapitre 2

Utilisation

2.1 menu

Pour utiliser le menu, cliquez sur le bouton droit dans la fenêtre de menu et relâchez sur l'item de votre choix.

Une fois qu'un choix a été fait dans le menu, il est conservé tant que l'on n'a pas fait d'autre choix. Par exemple, si on choisi « définition d'un point » dans le menu (ceci permet de cliquer sur un point pour avoir ses informations), on peut cliquer sur plusieurs points à la suite sans avoir à retourner dans le menu.

2.2 Zoom

Pour faire un zoom, il faut passer dans le mode zoom, ceci se fait en cliquant sur « ZOOM » dans le menu. Dans la fenêtre de commentaire, une aide au zoom s'affiche. Pour faire un zoom, il faut enfoncer, presser le bouton gauche de la souris sur le coin supérieur gauche du rectangle de zoom que l'on désire et relâcher le bouton sur le coin inférieur droit. Pour revenir au zoom précédent, il faut choisir « ZOOM ARRIERE » dans le menu. Pour arrêter le zoom, il faut choisir « FIN ZOOM » dans le menu, la fenêtre de commentaire nous indique que l'on a quitté le mode zoom (elle n'affiche plus d'aide au zoom).

Tant que l'on n'a pas quitté explicitement le mode zoom (clic sur Fin Zoom) ou effectué un choix dans le menu modifiant l'utilisation de la souris, on reste dans le mode zoom.

2.3 en général

En règle générale, il suffit de suivre les commentaires qui sont affichés dans la fenêtre de commentaire pour avoir plus de précision.

Chapitre 3

Aide à la programmation

Ce chapitre contient une série de conseils sur la manière d’incrémenter l’interface graphique du logiciel prplast2. Il est composé des sections suivantes : philosophie de programmation, création d’une nouvelle “boîte” , explication sur la programmation des points récurrent (zoom, écriture de commentaire, cadrage ...), et enfin quelques conseils/mise en garde.

Quelques principes d’openGL sont abordés mais nous resterons superficiels et il serait préférable d’utiliser en plus de la documentation openGL disponible très facilement sur Internet, en anglais ou en français ; ou encore de regarder dans les nombreux livres traitant d’openGL.

3.1 Makefile

utilisation

Le Makefile fourni permet de compiler l’ensemble du logiciel prplast2 sous Linux. Il est possible de trouver des équivalents pour windows.

Pour compiler le logiciel en entier taper la commande :
...>make

Ceci aura pour effet de recompiler tous les fichiers sources non à jour. C’est à dire ceux qui ont une date de modification postérieure à celle de l’exécutable et/ou du fichier binaire translatable (.o) correspondant. Ceci permet d’éviter de tout recompiler à chaque fois.

Attention !!! Il faut cependant faire attention car la gestion des dépendances n'est pas toujours parfaite. Ainsi dans certains cas il est nécessaire de tout recompiler. Par exemple, suite à la modification d'une interface (le nom d'une subroutine par exemple), le compilateur ne peut pas savoir que l'ancienne interface n'existe plus.

Pour recompiler la totalité du logiciel :

1. effacer : il s'agit d'effacer tous les fichier « .o », « .mod » et l'exécutable, ceci grâce à la commande
`...>make clean`
2. recompiler : il suffit de compiler normalement : commande unix
`...>make`
.

modification

!!!!!!ATTENTION!!!!!! La syntaxe des Makefile est très strict, surtout au niveau des tabulations, des espaces, des retours à la ligne.

Ce makefile compile uniquement des fichiers .f et .f90.

Le makefile crée les binaires translatables (.o) nécessaires à la compilation de l'exécutable, puis crée l'exécutable . Pour créer un fichier « xxx.o », il compile avec l'option « -c » le fichier « xxx.f90 » (fichier de nom identique sans l'extension). Si le fichier « xxx.f90 » n'existe pas il compile avec l'option « -c » le fichier « xxx.f ». Si ce dernier n'existe pas il renvoie une erreur.

Modifier le makefile consiste en fait à rajouter le(s) fichier(s) « .o » que l'on veut créer et indiquer qu'ils sont nécessaires à la compilation de l'exécutable.

ajout d'un module : Pour ajouter un module au projet (comme init, plast2 ...) il faut le rajouter dans la variable : **MOD_INCLUDE**, ce qui permet qu'il soit compilé en premier.

Pour rajouter le module dans la variable, il faut le mettre à la fin de la ligne en ajoutant un espace avant.

Par exemple si on veut rajouter le module **xxx** qui se trouve dans le fichier **xxx.f90** **MOD_INCLUDE = init.o** devient **MOD_INCLUDE = init.o xxx.o**

Pour ajouter un autre fichier, il faut procéder de la même manière que pour l'ajout d'un module, il suffit modifier la bonne variable.

Liste des variables :

- **MOD_INCLUDE** : contient les modules utilisés par les différentes boîtes
- **MODULES** : idem **MOD_INCLUDE** ...
- **OBJECTS** : contient tous les fichiers « .o » à compiler qui ne sont pas des modules et qui ne contiennent pas de modules utilisés dans d'autres fichiers. Cette variable contient de nombreux fichiers, ils sont donc rangés par ordre alphabétique, et un seul fichier par ligne. Le symbole « \ » permet d'ignorer le retour à la ligne. Ainsi les mots sont bien séparés par un seul espace. Donc si vous rajouter un fichier « .o » dans cette variable,
 1. positionnez vous sur la bonne ligne (ordre alphabétique)
 2. créez une nouvelle ligne
 3. rajoutez : **xxx.o **, en commençant au début de la ligne, avec uniquement un espace avant le symbole « \ ».
 4. En cas de problèmes : vérifiez que toutes lignes sont du type **xxx.o **, qu'il n'y ait pas d'autres espaces que celui obligatoire sur chaque, vérifiez aussi qu'il n'y ai pas de tabulations.
- **MOD_USED** : les fichiers sont compilés dans l'ordre ou il apparaissent dans la variable **OBJECTS**.
 Ils sont pour l'instant dans l'ordre alphabétique. Ainsi, si un module **m2**, dans un fichier **f2** est utilisé par un fichier **f1** alors le compilateur va renvoyer une erreur car il veut que le module **m2** soit compilé avant d'être utilisé.
 C'est pourquoi dans ce cas on place le fichier **f2.o** dans la variable **MOD_USED**, pour qu'il soit compilé avant les autres fichiers.
- **MOD_MOD** : cette variable contient les fichiers .mod à supprimer lors que l'appel à la commande **make clean**. Rajoutez ici tous vos modules : pour le module **xxx**, rajoutez **xxx.mod**.

3.2 philosophie de programmation

Cette section contient quelques points qui sont importants à avoir en tête lors de la modification du logiciel.

3.2.1 callbacks

Un callback ou fonction de rappel est une fonction, procédure ou sous-routine qui est appelée suite à un événement. Un événement pouvant être un clic avec la souris, le déplacement de la fenêtre, son rafraîchissement, etc. OpenGL utilise ce principe. Ainsi pour chaque événement que l'on veut traiter, il faut au préalable attacher un callback sur la fenêtre susceptible de traiter l'événement. Prenons pour exemple l'événement suivant : « l'utilisateur clic sur la fenêtre principale ».

1. tout d'abord il faut se positionner sur la fenêtre principale.

```
subroutine init
...
call glutSetWindow(imainWin)
...
end subroutine init
```

2. ensuite on attache la sous-routine de callbacks qui sera appelée lorsque l'utilisateur cliquera sur la fenêtre principale.

```
subroutine gestionSouris(button, state, x, y)
! les paramètres de cette sous-routine
! sont fixés par \opengl
integer(glcint), intent(inout) :: &
& button, state, x, y
...
! code exécuté à chaque clic de souris
! dans la fenêtre principale.
! En fait à chaque clic,
! cette méthode est appelée 2 fois
! car \opengl fait la différence entre
! l'enfoncement et le relâchement du bouton
! pour plus de détails, reportez vous à
! la section gestion de souris
...
end subroutine gestionSouris
```

```

subroutine init
  ...
  !on travail sur la fenêtr principale
  call glutSetWindow(imainWin)
  !attachement du callback gestionSouris
  call glutmousefunc(gestionSouris)
  ...
end subroutine init

```

3.3 Création d'une boîte

Cette section décrit le protocole à suivre pour créer une « boîte ». Il s'agit de créer une boîte à partir de rien. Cependant nous expliqueront en parallèle la mise à jour d'une boîte existante.

Cette section s'appuiera sur un exemple de boîte (simplifiée) : **piece**, qui affiche un body, et permet d'afficher son contour, de cliquer sur un point pour afficher ses informations, d'effectuer des actions de zoom et de revenir au menu précédent ou menu appelant.

3.3.1 changements effectués depuis la version GKS

Ce premier point est consacré aux modifications qui ont été apportées par rapport à l'ancienne version du logiciel (avant le passage à OpenGL).

- NUMPI,NOPIECE,NELPI : ces variables ont été mises dans la module plast2, il ne faut donc plus les passer en paramètre mais utiliser le module plast2.
- WXMIN,WYMIN,WXMAX,WYMAX : ces variables ont été mises dans le module varaffichage. Il n'est pas nécessaire d'utiliser des variables intermédiaires (WXINF, WXSUP, WYINF, WYSUP).
- ZOOM1 : l'appel de zoom1(WXINF,WXSUP,WYINF,WYSUP) a été supprimé, il faut appeler la subroutine **CADRE()**, qui se charge de recadrer la fenêtre d'affichage avec les valeurs de WXMIN, WYMIN, WXMAX, WYMAX.
- segment GKS : il n'y pas de notion de segment dans OpenGL, il y a uniquement un découpage de la fenêtre affichée en 2 sous fenêtres (commentaire et menu), se positionnant par dessus la fenêtre principale (ou de dessin).

3.3.2 INCLUDE

Tout d'abord, dans l'ancienne version du logiciel, écrit en fortran 77, les variables communes à plusieurs sous-routines (boîtes) étaient déclarées dans des fichiers .SDD, ces fichiers étant inclus si besoin. Dans la version actuelle ces fichiers ont été remplacés par des modules de la manière suivante :

INIT.SDD :

```

C_____
C STRUCTURE DE DONNEE INIT.SDD
C_____
C_____
C Initialisation de toutes les
C constantes utilisées
C dans le programme
C Initialisation du type
C des variables utilisées
C_____
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
REAL*4 PI,ZERO,UN,DEUX,TROIS,QUATRE,CINQ,SIX
PARAMETER (PI=3.141592654E+00)
PARAMETER (ZERO=0.E+00,UN=1.E+00,DEUX=2.E+00, &
&      TROIS=3.E+00,QUATRE=4.E+00, &
&      CINQ=5.E+00,SIX=6.E+00)
C_____
C      FIN DE SDD
C_____

```

devient **init.f90** :

```

module init
!_____
!Initialisation de toutes les
!constantes utilisées
!dans le programme
!_____
!le type des variables implicite doit
!être mis dans chaque nouveau module
!et/ou sous-routine
!IMPLICIT DOUBLE PRECISION (A-H,O-Z)
REAL*4 PI,ZERO,UN,DEUX,TROIS,QUATRE,CINQ,SIX
PARAMETER (PI=3.141592654E+00)

```

```

PARAMETER (ZERO=0.E+00,UN=1.E+00,DEUX=2.E+00, &
&          TROIS=3.E+00,QUATRE=4.E+00, &
&          CINQ=5.E+00,SIX=6.E+00)
end module init

```

Dans le fichier « xxx.f90 » qui utilise une variable contenue dans **init**, il faut remplacer

```

INCLUDE 'INIT.SDD'

par

USE INIT

```

Le fichier « GKSGKS.SDD » a été remplacé par le module « varaffichage » , dans le fichier « varaffichage.f90 » .

3.3.3 organisation

Toutes les « boîtes » créées jusqu'alors, ont une même organisation. Si l'on prend un fichier « xxx.f90 » (ou .f selon votre préférence, j'ai pour ma part utilisé .f90 pour une raison de compilateur et aussi pour signaler qu'on fait du fortran 90... A vous de voir cela ne change rien.), il est composé de :

- un module : **module callbacksXxx** qui contient le coeur de la « boîte » xxx.
- une subroutine : **subroutine xxx** qui permet de faire l'initialisation (cf. ci-après).

Dans notre exemple, cela donne :

```

module callbacksPiece
...
end module callbacksPiece

subroutine PIECE
...
end subroutine PIECE

```

3.3.4 initialisation

C'est la première partie constituant une boîte. Nous ne tenons pas compte des initialisations inhérentes au corps de la boîte (initialisation de variables ...).

Cette partie remplace dans les anciennes boîte l'appel à la subroutine MENU :

CALL MENU(ICH0IX,NBCHOIX,IA,STR)

L'initialisation se décompose en deux sous partie :

création du menu

la création du menu se fait en deux étapes :

1. sauvegarde du menu de l'appelant : Ceci permet de pouvoir restaurer le menu de l'appelant lorsque l'on quitte la boîte. On stock l'identifiant du menu de l'appelant dans une variable du module, car elle est persistante, c'est à dire qu'elle n'est pas détruite après l'exécution de la subroutine d'initialisation. Si on a une boîte « xxx » , par convention la variable qui va contenir le menu de l'appelant est nommé « iMenuAppelantXxx ».

```
!sauvegarde de l'identifiant du menu
!on veut travailler sur la fenêtre de Menu,
!il faut donc la positionner comme
!fenêtre courante
call glutSetWindow(imenuWin)
!on stock l'identifiant du menu courant
!dans la variable imenuAppelantPiece
imenuAppelantPiece = glutGetMenu()
```

2. appel la subroutine MENU qui prend en paramètre :
 - nbChoix : le nombre de choix que contient le menu
 - menuAppelé : le callback qui sera appelé quand l'utilisateur fera un choix dans le menu. Cette subroutine prend obligatoirement un seul paramètre : un entier qui contiendra la valeur du choix de l'utilisateur.
 - IA : le tableau d'entier contenant la longueur de chaque élément du menu
 - STR : la chaîne de caractère contenant les éléments du menu

Dans notre exemple cela donne :

module callbacksPiece

Integer :: imenuAppelantPiece

contains

```
subroutine menuPiece(iChoix)
  integer , intent(in) :: iChoix
```

```

    if (ichoix==1) then
        !calcul le contour
        call contour()
        !dessine le contou
        call desContour()
        ...
    endif
end subroutine menuPiece

end module callbacksPiece

SUBROUTINE PIECE
    !contient la variable imenuAppelantPiece
    use callbacksPiece
    use opengl_glut
    ...
    Integer, DIMENSION(1:20) :: IA
    CHARACTER*256 STR

    integer :: ICHOIX,NBCHOIX

    NBCHOIX=6
    STR='CONTOURDEF POINTZOOMZOOM ARRIEREFIN ZOOMFIN'
    IA(1)=7
    IA(2)=9
    IA(3)=4
    IA(4)=12
    IA(5)=8
    IA(6)=3

    ...
    !-----initialisation
    !sauvegarde de l'identifiant du menu
    call glutSetWindow(imenuWin)
    imenuAppelantPiece = glutGetMenu()

    !création du menu de piece
    call menu(nbChoix,menuPiece,IA,STR)

    ...
    !-----Fin initialisation

```

END SUBROUTINE PIECE

attachement des callbacks

Comme nous l'avons vu dans la section 3.2.1, il s'agit d'attacher un callback pour tous les événements que l'on veut traiter. En général, on traite principalement deux événements : le réaffichage de la fenêtre de commentaire et celui de la fenêtre principale. Nous traiterons l'évènement « l'utilisateur clic dans la fenêtre principale » plus tard dans cette section.

Dans notre exemple : on affiche le body sur lequel on travail et on écrit dans la fenêtre commentaire son numéro, on modifie le callbacks de la fenêtre principale et celui de la fenêtre de dessin.

module callbacksPiece

...

contains

subroutine mainDisplayPiece()

CALL DESPIECE()

end subroutine mainDisplayPiece

subroutine comDisplayPiece()

character :: TEXT*80,TEXT6*6

integer :: NOZODI

!permet d'effacer la fenêtre ,

!il faut appeler cette méthode

!à chaque fois si l'on ne veut pas

!avoir un résultat très étrange

call efface('com')

!dessine les contours de la sous fenêtre en bleu

call desContourFenetre('com')

NOZODI=2

WRITE(TEXT6,'(I6) ') NUMPI

TEXT= 'DESSIN du BODY num' // TEXT6

!écrit le TEXT en blanc à la ligne

!NOZODI de la fenêtre de commentaire

CALL ECTX(NOZODI,TEXT)

end subroutine comDisplayPiece

```

end module callbacksPiece

SUBROUTINE PIECE
  use opengl_glut
  ...

  !on affiche la pièce numéro
  !dans notre exemple
  NUMPI=1
  !(NUMPI se trouve dans le module plast2)

  !pour pouvoir afficher la pièce
  !on a besoin de cadrer
  !pour cela on calcul les points
  !bas, gauche (WXMIN,WYMIN)
  !et haut, droit (WXMAX,WYMAX)
  !de la pièce
  call rechEspaceMin()
  !puis on cadre avec les points précédents
  call cadre()

  !-----initialisation
  ...
  !modification des callbacks d'affichage
  !pour la fenêtre principale
  call glutSetWindow(icomWin)
  call glutdisplayfunc(comDisplayPiece)
  !on réaffiche la fenêtre
  !une fois qu'elle est initialisée
  call glutPostRedisplay()

  !pour la fenêtre de commentaire
  call glutSetWindow(imainWin)
  call glutdisplayfunc(mainDisplayPiece)
  call glutPostRedisplay()
  !-----Fin initialisation
END SUBROUTINE PIECE

```

Aller voir la section 3.5.3 pour avoir des précisions sur les callbacks de réaffichage des fenêtres.

En fait la subroutine PIECE est maintenant terminée, il s'agit d'écrire

les callbacks. La subroutine `PIECE` sert uniquement à l'initialisation.

3.3.5 callbacks

affichage

Il est important de garder à l'esprit à quel moment une fenêtre est rafraîchie. Ceci impose certaines contraintes dans la programmation.

Notamment, cela oblige à attacher un callback pour un affichage qui doit durer.

En effet la fenêtre se réaffiche quand vous lui demandez (voir ci-après), quand elle retrouve le focus (c'est à dire qu'elle n'était pas en premier plan et repasse en premier plan, suite à un clic par exemple), quand elle est déplacé, ...

Ainsi si on ne met pas de callback pour afficher quelque chose, cela peut très bien disparaître sans qu'on l'ai désiré : si l'utilisateur va par exemple faire une saisie dans la console.

Cependant si on peut éviter d'utiliser les callbacks, cela permet de gagner du temps et de la mémoire. Par exemple, quand l'utilisateur doit faire 2 clic à souris pour définir le 1^{er} et le 2^e, si on veut afficher le 1^{er} point, il n'est pas nécessaire de tout réafficher. Cependant, si jamais la fenêtre est réaffichée le point ne le sera pas et il faut avertir l'utilisateur qu'il en est au deuxième point, dans le callbacks de la fenêtre de commentaire .

gestion de la souris

`openGL` permet de traiter les clics de souris dans une fenêtre. Pour interpréter ces événements, il faut attacher à la fenêtre en question un callback de gestion des événements souris. Ce dernier sera appelé lorsque l'un des 3 boutons de la souris est pressé et/ou relâché dans la fenêtre. Un callback de gestion de souris est une subroutine avec les paramètres suivants :

- `ibutton` : c'est un **entier** qui représente le bouton de la souris qui est cliqué, il y a donc 3 possibilités :
 - `GLUT_LEFT_BUTTON`
 - `GLUT_MIDDLE_BUTTON`
 - `GLUT_RIGHT_BUTTON`
- `istate` : c'est un **entier** qui représente l'état du bouton :

- **GLUT_DOWN** si le bouton est enfoncé
- **GLUT_UP** si le bouton est relâché.
- ix : c'est un **entier** qui représente l'abscisse du curseur au moment où l'évènement est déclenché. Attention ceci est en coordonnées relatives, c'est à dire par rapport au coin supérieur gauche de la fenêtre.
- iy : c'est un **entier** qui représente l'ordonnée du curseur au moment où l'évènement est déclenché. Attention ceci est en coordonnées relatives.

Attention, si on clic (pression + relâchement d'un bouton) dans la fenêtre (ayant un callbacks de gestion souris), alors le callbacks sera appelé 2 fois.

```
subroutine sourisPiece(button,state,x,y)
  use opengl_gl
  ...
  integer(gl Cint), intent(inout) :: button, state, x, y
  ...
end subroutine sourisPiece
```

```
subroutine xxx
  use opengl_glut
  ...
  call glutSetWindow(imainWin)
  call glutmousefunc(sourisPiece)
  ...
end subroutine xxx
```

Pour détacher un callback d'une fenêtre, il faut passer un pointeur de fonction « null » à la subroutine « glutmousefunc ». En fait on attache un callback null. Dans la librairie f90gl, le pointeur de fonction null est : **GLUTNULLFUNC**.

```
subroutine xxx
  use opengl_glut
  ...
  call glutSetWindow(imainWin)
  call glutmousefunc(GLUTNULLFUNC)
end subroutine xxx
```

Pour plus de précision sur les callbacks de gestion souris, reportez vous à la section 3.4.6 page 33.

3.3.6 remplissage de la boîte

Rendez vous à la section 3.4, page 26 pour avoir des informations sur ce qui compose en générale une boîte (gestion souris, menu, choix des couleurs, sous programme de dessin ...).

3.3.7 fin, retour au menu supérieur

Avant de quitter une boîte, il faut :

1. rattacher les callbacks d’affichage de l’appelant. Dans notre exemple de pièce, nous avons dans l’initialisation modifié les callbacks d’affichage de la fenêtre principale et de la fenêtre de commentaire. Ainsi nous devons restaurer ceux de l’appelant pour chacune de ses fenêtres. Pour pouvoir connaître les callbacks de l’appelant, il faut utiliser le module « `callbacksAppelant` ».

```
...
use callbacksAppelant
...

!fenêtre de commentaire
call glutSetWindow (icomWin)
!attachement du callback
!d’affichage de l’appelant
call glutdisplayfunc (comDisplayAppelant)
!on ne traite plus les clics
call glutmousefunc (GLUTNULLFUNC)
!on raffiche la fenêtre pour
!afficher le bon commentaire
call glutPostRedisplay ()

!fenêtre principale
call glutSetWindow (imainWin)
call glutdisplayfunc (mainDisplayAppelant)
call glutmousefunc (GLUTNULLFUNC)
call glutPostRedisplay ()
```

2. détacher les éventuels callbacks de gestion de souris. Pour annuler la gestion des événements souris sur une fenêtre il faut passer un pointeur de fonction null (GLUTNULLFUNC car nous utilisons la librairie f90gl) lors de l’attachement d’un callback de gestion de souris.

```
call glutSetWindow (imainWin)
call glutmousefunc (GLUTNULLFUNC)
```

3. détruire le menu de la boîte. Pour éviter une utilisation néfaste de la mémoire, il faut détruire le menu courant, car il ne sera plus utilisé par la suite : on crée un nouveau menu à chaque fois.

```

!on se met sur la fenêtre de menu
call glutSetWindow (imenuWin)
!on ne peut supprimer le menu courant
!que s'il existe (évite les crashes ...)
if( glutGetMenu() /= 0 ) then
    !le menu était attaché
    !au bouton droit de la souris ,
    !avant de la supprimer ,
    !il faut le détacher de ce bouton
    call glutDetachMenu(GLUT_RIGHT_BUTTON)
    !destruction du menu
    call glutDestroyMenu(glutGetMenu())
end if

```

4. restaurer le menu de l'appelant. Pour restaurer le menu de l'appelant, il suffit de le fixer dans la fenêtre courante.

```

!on doit être sur la fenêtre de menu
...
!le menu de l'appelant
!n'a pas été détruit , donc il suffit
!de le fixer à la fenêtre
call glutSetMenu(imenuAppelantPiece)
!on attache le menu courant ,
!qui est maintenant celui de
!l'appelant sur le bouton droit de la souris
call glutAttachMenu(GLUT_RIGHT_BUTTON)

```

Maintenant que vous avez fait votre boîte, il ne vous reste plus qu'à l'inclure dans le Makefile pour qu'elle se compile avec le reste du logiciel, et que vous puissiez ainsi l'utiliser. Pour savoir comment faire , reportez vous à la section 3.1, page 13.

3.4 Explication des points récurrents

3.4.1 cadre

Pour recadrer la fenêtre d'affichage, il faut modifier les valeurs de WXMIN, WYMIN, WXMAX et WYMAX puis appeler la subroutine cadre().

```

WXMIN = 0
WYMIN = 0

```

```

WXMAX = 10
WYMAX = 20
CALL CADRE()
!la fenêtre est maintenant recadrée :
!son coin inférieur gauche a pour coordonnées : (0,0)
!son coin supérieur droit a pour coordonnées : (10,20)

```

3.4.2 couleurs

OpenGL utilise une variable d'état pour la gestion de la couleur. C'est à dire que tous les dessins sont faits dans la couleur « courante ». Ainsi pour changer de couleur, il faut d'abord charger la couleur que l'on désire puis dessiner ce que l'on veut, sans préciser la couleur. Un module (**couleur**) a été fait pour rendre plus lisible la gestion des couleurs. Pour changer de couleur il faut appeler la subroutine choixCouleur(kcoul) qui prend en paramètre un entier. Le tableau 3.1 établit la correspondance entre kcoul et la couleur.

KV1	blanc
KV2	rouge
KV3	jaune
KV4	vert
KV5	cyan
KV6	bleu
KV7	magenta
KV8	gris clair
KV9	orange
autre	blanc

FIG. 3.1 – tableau de correspondance valeur -> couleur

Les valeurs KV1,...,KV9 sont des constantes entières définies telles que : KV1=1, KV2=2, ... , KV9=9.

équivalent GKS à la méthode choix couleur : CALL GSPLCI(kcoul)

3.4.3 dessin

Avant toute chose, il faut effacer la fenêtre, cela se fait grâce à l'appel de la méthode efface avec « main » en paramètre pour indiquer que l'on veut effacer la fenêtre principale.

```
!effacement de la fenêtre de dessin
call efface ('main')
```

forme géométrique

Pour dessiner une forme dans la fenêtre courante :

```
!on dit ce qu'on va dessiner : ici un point
call glBegin (GL_POINTS)
!on donne les sommets de ce que l'on veut dessiner
!ici il s'agit uniquement d'un point on a donc un
!sommet (pour dessiner une ligne il en faudrait 2,
!un carre 4, etc ...).
call glVertex3d (X,Y,0.0_gldouble)
!on finit le dessin de la forme
call glEnd ()
```

glVertex3d

- **gl** signifie que la subroutine appartient au module `openGL_gl`, faisant le lien avec la bibliothèque en langage C **libgl**, c'est le même principe pour les subroutines des librairies `glut` et `glu`.
- **Vertex** représente un sommet
- **3** signifie que l'on dessine dans un repère x,y,z
- **d** signifie que les coordonnées sont données par un réel de type **gldouble** avec double précision, pour un réel simple utilisez **f**. Pour écrire une valeur constante sous la forme d'un « `gldouble` », pour éviter une erreur de compilation lors de l'appel à « `glVertex3d` », il faut rajouter **_gldouble** après le réel. Par exemple : « **9.5_gldouble** ».

liste non exhaustive des formes à dessiner

Pour avoir des images du code fournis vous pouvez vous rendre sur le site suivant : <http://raphaello.univ-fcomte.fr/IG/OpenGL/OpenGL-1.htm>

- **tracé du contour d'un quadrilatère** : avec `openGL` on a besoin des 4 sommets (dont les coordonnées sont : `XTOGL(1), YTOGL(1), ... XTOGL(4), YTOGL(4)`). On a donc pas besoin d'un cinquième sommet, égal au premier, comme c'était le cas pour `GKS`.

```
!GL_LINE_LOOP permet de rajouter un trait
!entre le dernier et le premier point
call glBegin (GL_LINE_LOOP)
```

```

call glVertex3d (XTOGL(1),YTOGL(1),0.0 _gldouble)
call glVertex3d (XTOGL(2),YTOGL(2),0.0 _gldouble)
call glVertex3d (XTOGL(3),YTOGL(3),0.0 _gldouble)
call glVertex3d (XTOGL(4),YTOGL(4),0.0 _gldouble)
call glEnd ()

```

équivalent *GKS* : *CALL GPL(KV5,XTGKS,YTGKS)*, avec 5 points dans *XTGKS* et *YTGKS*, le dernier égal au premier

- **tracé d'un quadrilatère plein** : On a la encore besoin de 4 sommets, et on dessine un **GL_QUADS** avec quatre sommets :

```

call glBegin (GL_QUADS)
call glVertex3d (XTOGL(1),YTOGL(1),0.0 _gldouble)
call glVertex3d (XTOGL(2),YTOGL(2),0.0 _gldouble)
call glVertex3d (XTOGL(3),YTOGL(3),0.0 _gldouble)
call glVertex3d (XTOGL(4),YTOGL(4),0.0 _gldouble)
call glEnd ()

```

- **tracé d'une ligne** : On peut tracer une ligne entre 2 sommets (ou plus, les lignes se faisant dans l'ordre d'apparitions des sommets). On dessine un **GL_LINES** :

```

call glBegin (GL_LINES)
call glVertex3d (XTOGL(1),YTOGL(1),0.0 _gldouble)
call glVertex3d (XTOGL(2),YTOGL(2),0.0 _gldouble)
call glEnd ()

```

équivalent *GKS* : *CALL GPL(KV5,XTGKS,YTGKS)*, avec 2 points dans *XTGKS* et *YTGKS*

- **tracé d'un point** : On peut tracer un ou plusieurs point(s), à l'aide de **GL_POINTS** :

```

call glBegin (GL_POINTS)
call glVertex3d (XTOGL(1),YTOGL(1),0.0 _gldouble)
call glEnd ()

```

équivalent *GKS* : *CALL GPL(KV5,XTGKS,YTGKS)*, avec 1 point dans *XTGKS* et *YTGKS*. De plus, on a pas à choisir la forme du point comme dant *GKS*

- **tracé d'un triangle** : On peut tracer un triangle, à l'aide de **GL_TRIANGLES** :

```

call glBegin (GL_TRIANGLES)
call glVertex3d (XTOGL(1),YTOGL(1),0.0 _gldouble)
call glVertex3d (XTOGL(2),YTOGL(2),0.0 _gldouble)
call glVertex3d (XTOGL(3),YTOGL(3),0.0 _gldouble)

```

```
call glEnd ()
```

équivalent GKS : CALL GFA(KV3,XTGKS,YTGKS), avec 3 points dans XTGKS et YTGKS.

mise à jour

Les points cités précédemment s'appliquent bien entendu. Quelques modification supplémentaire sont à apporter. Les appels GKS sont à remplacer évidemment (ceci est expliqué dans ce qui suit, pour chaque appel présenté, on donnera son équivalent GKS en italique). Pensez aussi à utiliser des « gldouble » pour XTOGL, et YTOGL, pour éviter de faire hurler le compilateur.

```
real(kind=gldouble) :: XTOGL(4),YTOGL(4)
```

modification de l'épaisseur d'une ligne, d'un point

On peut modifier l'épaisseur d'une ligne ou d'un point, le nombre de pixels utilisés pour représenter ce point (ou les points de la ligne). Par défaut, la taille d'un point est fixé à 1 pixel.

```
!les prochains points affichés  
!seront sur 3 pixels  
call glPointSize(3)  
!les points des prochaines lignes affichées  
!seront sur 3 pixels  
call glLineWidth(3)
```

Vous trouverez dans le tableau 3.2 un récapitulatif des changement à faire pour passer des appels GKS aux appels OpenGL. On ne mentionnera que le type d'objet à dessiner (défini par glBegin(...)) car le reste est identique.

objet à dessiner	appel GKS	appel openGL
contour d'un carré	GPL(KV2,XTOGL,YTOGL)	glBegin(GL_LINE_LOOP)
quadrilatère plein	GPL(KV2,XTOGL,YTOGL)	glBegin(GL_QUADS)
contour d'un triangle	GPL(KV2,XTOGL,YTOGL)	glBegin(GL_LINE_LOOP)
triangle plein	GPL(KV2,XTOGL,YTOGL)	glBegin(GL_TRIANGLES)
ligne	GPL(KV2,XTOGL,YTOGL)	glBegin(GL_LINES)
point	CALL GPM(KV1,XTOGL,YTOGL)	glBegin(GL_POINTS)
action à faire	appel GKS	appel openGL
choix couleur KV1	CALL GSPLCI(KV1)	call choixCouleur(KV1)
épaisseur d'un point	CALL GSMKSC(UN)	call glPointSize(UN)
épaisseur d'une ligne	CALL GSLWSC(UN)	call glLineWidth(UN)
représentation du point	CALL GSMK(KV3)	un point est un point

FIG. 3.2 – tableau de correspondance GKS -> openGL

3.4.4 écriture de texte

écriture dans la fenêtre de commentaire

Pour écrire un texte blanc d'une ligne entière dans la fenêtre de commentaire, utilisez la méthode `ECTX(NOZODI,TEXT)`. **NOZODI** est un entier indiquant sur quel ligne le texte **TEXT** on écrit.

```
CALL ECTX(2,'écrit sur la ligne 2 en blanc')
```

Pour écrire un texte en couleur d'une ligne entière dans la fenêtre de commentaire, utilisez la méthode `ectxCouleur(nozodi,txt,kcoul)`.

```
call ectxCouleur(2,'écrit sur le ligne 2 en rouge ',KV2)
```

Pour écrire un texte en couleur d'1/5 de ligne dans la fenêtre de commentaire, utilisez la méthode `ectxColoneCouleur(nLigne,nCol,txt,kcoul)`.

```
!écrit le texte en rouge
```

```
!à la ligne 3 et
```

```
!dans la colonne 2
```

```
call ectxColoneCouleur(3,2,'13 col 2',KV2)
```

écriture dans la fenêtre principale

Pour écrire le texte `txt` en couleur dans la fenêtre principale avec pour coordonnées `(xt,yt)` (coordonnées dans le repère de base), utilisez la méthode `ectxMainCouleur(txt,xt,yt,kcoul)`.

Vous pourrez trouver le corps des méthodes d'écriture se trouve dans l'annexe D.2, page 60.

3.4.5 menu

Lorsque l'utilisateur effectue un choix dans le menu, le callbacks de menu est appelé. Reportez vous à la section 3.3.4, page 19 pour savoir comment attacher un callbacks de menu à une fenêtre. Quand le callbacks est appelé, le paramètre `iChoix` a la valeur de l'item sélectionné dans le menu. Ainsi pour savoir quelle action effectuer, il suffit de faire un test sur le paramètre `iChoix`. Dans notre exemple voilà ce qu'on obtient :

```
subroutine menuPiece(iChoix)
  integer, intent(in) :: iChoix
  if (iChoix==1) then
    call contour()
  elseif (iChoix==2) then
```

```

    ...
endif
end subroutine menuPiece

```

3.4.6 souris

Comme nous l'avons vu dans la section 3.3.5, le callback de gestion de souris est appelé quand l'utilisateur presse et/ou relâche un des 3 boutons dans la fenêtre. Les coordonnées du point cliqué sont en coordonnées relatives, il faut donc les convertir en coordonnées réelles. Il faut ensuite appliquer les transformations inverses sur ces nouvelles coordonnées pour obtenir les coordonnées du point dans le repère de base.

```

subroutine sourisInfoPoint(button,state,x,y)
  use varAffichage
  integer(glcint), intent(inout) :: button, state, x, y
  DOUBLE PRECISION xreel,yreel,xcr,ycr,xcw,ycw
  character :: text*80

```

```

  !on ne traite que l'enfoncement du bouton gauche
  if(state == GLUT_DOWN .AND. button == &
&      GLUT_LEFT_BUTTON ) then

```

```

    !centre de la fenêtre
    !(en tenant compte des sous fenêtres)
    !ces coordonnées sont réelles,
    !dans le repère écran.
    XCW=(-0.99+0.50)/2.
    YCW=(0.99-0.50)/2.

```

```

    !centre de la figure en coordonnées réelles,
    !dans le repère de base
    XCR=(WXMIN+WXMAX)/2.
    YCR=(WYMIN+WYMAX)/2.

```

```

    !x et y sont en coordonnées relatives
    !(par rapport au coin supérieur gauche
    !de la fenêtre principale)
    !conversion des coordonnées en coordonnées réelles
    !dans le repère écran

```

```

xreel = real(x)/(WIDTH/2.)-1
yreel = real(y)/(HEIGHT/2.)-1

!transformation inverse des points
!pour avoir les coordonnées dans
!le repère de base:
!homotécie inverse + translation inverse
xreel = xreel*(rapport/0.80)-(XCW*RAPPORT-XCR)
yreel = -yreel*(rapport/0.80)-(YCW*RAPPORT-YCR)

!on va écrire dans la fenêtre de commentaire
call efface('com')
call desContourFenetre('com')

IF (XREEL<WXMIN.OR.XREEL>WXMAX &
      & .OR.YREEL<WYMIN.OR.YREEL>WYMAX) THEN
  write(6,*)'clic hors cadre'
else
  write(6,*)'coordonnées du point : '
  write(6,*)'X = ',xreel, ' ', Y = ', yreel

  !les points sont dessinés sur 3 pixels
  call glPointSize(TROIS)
  !les points sont dessinés en rouge
  call choixCouleur(KV2)
  !les points sont dessinés
  !dans la fenêtre principale
  call glutSetWindow(imainWin)
  !dessin du point
  call glBegin (GL_POINTS)
  call glVertex3d(xreel,yreel,0.0_gldouble)
  call glEnd()

  !on notera que ce point n'étant pas dessiner
  !dans une subroutine de callback ne sera pas
  !réaffiché avec la fenêtre
end IF
end if
end subroutine sourisInfoPoint

```

3.4.7 zoom

Pour pouvoir faire un zoom, un zoom arrière, arrêter de zoomer, il faut modifier le menu et le callback de menu :

Premièrement, il faut rajouter dans le menu 3 item : zoom, zoom arrière, fin zoom. En fait il s'agit de compléter la chaîne STR avant de l'appel à la subroutine MENU.(cf. section 3.3.4 page 19).

Si nous reprenons notre exemple de la pièce :

```

subroutine menuPiece(iChoix)
  use varAffichage
  integer, intent(in) :: iChoix

  if (ichoix==1) then
    ...
    !_____
    !  ZOOM
    !_____
  elseif(iChoix==3)then
    !on passe dans le mode zoom
    bzoom = .TRUE. !variable du module varAffichage
    call sauvZoom()
    !on affiche l'aide au zoom
    !dans la fenêtre de commentaire
    call glutSetWindow(icomWin)
    call glutdisplayfunc(comDisplayZoom)
    call glutPostRedisplay()

    call glutSetWindow(imainWin)
    call glutmousefunc(zoomSouris)

    !_____
    !  ZOOM ARRIERE
    !_____
  elseif(iChoix==4)then
    call restaurZoom()

    !on recadre
    call cadre()

    !on rafraîchit la fenêtre d'affichage

```

```

call glutSetWindow (imainWin)
call glutPostRedisplay ()

!_____
!  FIN ZOOM
!_____
elseif (iChoix==5)then
  !on sort du mode zoom
  bzoom = .FALSE.
  call remetInitZoom ()

  !on change l'affichage des commentaires
  call glutSetWindow (icomWin)
  call glutdisplayfunc (comDisplayPiece)
  call glutPostRedisplay ()

  !on recadre
  call cadre ()

  !on rafraîchit la fenêtre d'affichage
  call glutSetWindow (imainWin)
  call glutPostRedisplay ()

  ...
endif
end subroutine menuPiece

```

Vous trouverez Le corps des méthodes de zoom de trouve dans l'annexe D.1, page 56.

3.5 conseils, mise en garde

Cette section contient quelques mises en garde, permettant d'éviter des erreurs de programmation. Elle contient aussi quelques points sur le fonctionnement d'OpenGL, elle vient en complément de la section 3.2.

3.5.1 variables

Il est important de faire attention à la portée, visibilité, durée de vie des variables, surtout quand l'on crée une boîte à partir d'une boîte existante.

3.5.2 fenêtre courante

OpenGL utilise la notion de fenêtre courante, c'est à dire que toutes les actions sont effectuées dans la fenêtre courante. Par exemple, on veut modifier l'affichage de commentaire suite à un clic de l'utilisateur dans la fenêtre d'affichage.

Premièrement il faut fixer la fenêtre de commentaire comme étant la fenêtre courante puis écrire le texte, sinon il s'affichera dans la fenêtre principale.

3.5.3 rafraîchissement

Avant de redessiner une fenêtre, il est obligatoire de l'effacer, pour vider les tampons de couleurs, ceci permet de ne pas utiliser de la mémoire inutilement. Dans ce logiciel, une subroutine a été créée pour que cela soit plus lisible, ainsi pour effacer une sous fenêtre (commentaire, menu ou principale), il faut faire appel à la subroutine efface :

```
!pour la fenêtre de commentaire
call efface('com')
!pour la fenêtre principale ou d'affichage
call efface('main')
!pour la fenêtre de menu
call efface('menu')
```

Une autre subroutine a été créée pour dessiner le contour d'une sous fenêtre en bleu, pour pouvoir délimiter les sous fenêtres.

```
!pour la fenêtre de commentaire
call desContourFenetre('com')
!pour la fenêtre principale ou d'affichage
Cela n'apporte rien.
De plus pour pouvoir utiliser
la subroutine desContourFenetre, il
faudrait recadrer entre -1 et 1 ...
!pour la fenêtre de menu
call desContourFenetre('menu')
```

Pour demander le rafraîchissement d'une fenêtre, il faut utiliser la subroutine **glutPostRedisplay**. Ceci permet de noter la fenêtre comme étant à rafraîchir au prochain tour de la boucle d'affichage.

Il est conseillé d'utiliser cette méthode, plutôt que d'appeler directement le callback d'affichage.

Cependant dans certains cas, on a besoin d'un rafraîchissement immédiat. Par exemple pour une fenêtre de commentaire indiquant à l'utilisateur d'aller faire une saisie en console, immédiatement suivi d'un `read(5,*)`. Dans ce cas il faut uniquement quelques instruction machine pour que le `read(5,*)` soit fait, ce qui empêche le rafraîchissement de se faire à temps, (temps < quelques micros voir nanosecondes selon la puissance de l'ordinateur).

Annexe A

complément sur l'installation

A.1 glut3.7

A.2 f90gl

A.2.1 INSTALL

Voici le fichier INSTALL fournis par les développeurs de la librairie f90l, il contient des explication plus complète sur la marche à suivre pour installer la librairie

This file contains instructions for building and installing the f90gl libraries. Check the f90gl web page <http://math.nist.gov/f90gl> for the latest information, makefiles for additional systems, etc. Look under the list of Unix workstations or list of Windows 95/NT systems for the status of systems that don't work at the time of this release. Look under Software for additional makefiles. Look under News for general announcements.

Table of Contents :

1. GLUT
2. system code
3. fpp preprocessor file
4. makefile
5. compile and install
6. test
7. system specific notes

- all systems
- NASoftware
- Lahey
- DVF
- Salford
- XLF
- Digital Fortran on Digital Unix
- Pacific Sierra
- gfortran

1. GLUT

You need version 3.7 or later of GLUT. Build GLUT as per the usual instructions in the GLUT distribution, but be certain to copy both `glut.h` and `glutf90.h`, to an `include/GL` directory.

It seems the confusion of GLUT 3.7 versions will never be straightened out. There are early versions of GLUT 3.7 still around, and in some very popular places (like Red Hat rpms). These early 3.7 beta versions will not work with `f90gl`. The correct versions of GLUT will have `GLUT_XLIB_IMPLEMENTATION=14` or larger in `glut.h`. If you are not certain that you have the correct version of GLUT, you should obtain GLUT from the `f90gl` web page <http://math.nist.gov/f90gl>. It is available in `.tar.gz` and `.zip` formats, and precompiled for Windows.

The GLUT home page is <http://reality.sgi.com/opengl/glut3/glut3.html>

2. system code

Makefiles are provided for all systems on which `f90gl` has been tested. To keep the names within the 8.3 naming convention, a 4 (or 5) character code is assigned to each system configuration. These characters designate the hardware, operating system, Fortran compiler, OpenGL and (optionally) special designations. For example, `s4nm` is the code for a Sun workstation with SunOS 4.x, the NAGware compiler, and Mesa. Look in the file `mf_keys` to determine the code for your system.

If you do not find a preprocessor file and makefiles for your system, you should find a similar system and make your own files using the files for that system as a starting point. If you have success with a new system and would like to share your makefiles and preprocessor file with other users, send email to william.mitchell@nist.gov.

3. fpp preprocessor file

`f90gl` uses a simple preprocessor for conditional compilations to handle differences in the OpenGL version, supported extensions, ways that

Fortran and C interface, limitations in the subset languages, and compiler bugs.

The directory f90gl/fppincs contains files of definitions for the preprocessor for each system. The names of these files are of the form fppxxxx where xxxx is the system code described above. There is also a file fppnada which is used by systems that do not need any information in this file. As of version 1.2.6, the remainder of this section is relevant only to MS Windows systems, except for the last paragraph.

Identify the fpp file for your system, or create one from the file for a similar system. Then verify that the #define statements are correct for your system. There are three sets of these : OpenGL version, extensions, and special system designations.

You must first identify the OpenGL version and extensions supported by your OpenGL library. To do this, look at the file ???/include/GL/gl.h. To determine your OpenGL version, look for the string VERSION. There should be a few of them of the form GL_VERSION_1_1, GL_VERSION_1_2, etc. These tell you what version is supported.

The fpp file should have a #define for each version number up to your version of OpenGL. For example, if you have OpenGL 1.1 then the fpp file should have #define OPENGL_1_0 #define OPENGL_1_1 !#define OPENGL_1_2 Note that the last one is commented out.

To determine which extensions are supported by your OpenGL, search for GL_EXT in gl.h. You should find the list of supported extensions, e.g. /* Extensions */

```
#define GL_EXT_abgr 1
#define GL_EXT_blend_color 1
```

The fpp file has a #define statement for each extension currently supported by f90gl. You should comment out any that are not supported by your OpenGL. If you don't comment out one that is not supported, you will get undefined symbol errors from the linker. If you do comment out one that is supported, you will not be able to use that extension from Fortran. If your favorite extension is not listed in the fpp file, it is not currently supported by f90gl; send email to william.mitchell@nist.gov and we'll consider adding it to the next release.

The final set of #define statements indicate system configuration details, usually indicating what compiler is being used. These are only necessary when the f90gl source code needs to do something special for this system, for example work around a compiler bug. You should probably leave these alone, and for a new system not define any. An exception

is `GLU_1_1_TESS` which should be defined if and only if the OpenGL library does not support GLU 1.2 tessellation. Look for `GLU_VERSION_1_2` in `.../include/GL/glu.h`.

4. makefile

The makefile names begin with `mf`. In the top directory there is a file for each system on which `f90gl` has been tested of the form `mfxxxx` (for makefiles) or `mfxxxx.bat` (for DOS batch files) where `xxxx` is the system code.

Identify your makefile and edit it to set the directories in which your system keeps the OpenGL and X11 include files. If your system is not supported, then find a similar system and use it as a starting point for your makefile. You will probably have to change some additional macros at the beginning of the makefile.

The subdirectories `util`, `gl`, `glu` and `glut` contain makefiles that are called by the top level makefile. Some of these are common to many systems (e.g. `mfmost` for most systems), and some are system specific using the name `mfxxxx` where `xxxx` is the system code. You should not have to make any changes to these makefiles.

The subdirectory `examples` contains a makefile for each system on which `f90gl` has been tested. This will need to be edited to for your system similar to the top level makefile.

5. compile and install

If `xxxx` is your system code, then from the top `f90gl` directory :

at a Unix prompt, type `make -f mfxxxx` at a DOS prompt, type `mfxxxx`
exception : for the PGI compiler on Windows, use `make`

This will compile the `f90gl` libraries, move them to `f90gl/lib`, and move any "module" files to either `f90gl/include/GL` or `f90gl/lib`, depending on the system. After testing, you can either leave them there or move them to another home, for example the `lib` and `include` directories containing OpenGL.

"`make -f mfxxxx clean`" (in Unix) or "`mfxxxx clean`" (in DOS) will delete all files generated by the building process, but leave the library and module files in `f90gl/include/GL` and `f90gl/lib`.

"`make -f mfxxxx reallyclean`" or "`mfxxxx reallyclean`" will delete the files in `f90gl/include` and `f90gl/lib`, too.

6. test

Finally you should test the libraries by building and running the example programs in `f90gl/examples`. See the `README` file in that directory.

7. system specific notes

all systems

- no notes

NASoftware/Linux

- you must use FortranPlus Version 2.0.5 or later.
- you must have the directory containing the libraries (e.g. <path-to-f90gl>/f90gl/lib) in your LIBRARY_PATH environment variable to build the examples

NASoftware/WinNT

- you must use FortranPlus Version 2.0.55 or later

Lahey

- Lahey supplies a binary version of f90gl that works with LF90, LF95, and ELF90. Go to <http://www.lahey.com/andsearchforOpenGL>. The remaining comments pertain to building f90gl from the sources instead of using the precompiled libraries.
- Note that for Lahey/Fujitsu LF95 on Windows the mf key uses "j" for Fujitsu.
- you must use LF95 if you use MS VC++. With the Borland C compiler you can use either LF90 or LF95.
- with LF95 5.5 or later and MS VC++, you may find multiple definitions of WinMain when you link an example. If so, comment out the line that creates winmain.obj in f90gl/gl/mf8njo

DVF

- Compaq (Digital) supplies a binary version of f90gl for DVF. Go to [http://www.compaq.com/fortranandlookunder\"Downloads](http://www.compaq.com/fortranandlookunder\)".

Salford

- does not currently work with Salford ftn95. Check the f90gl web page for the latest information.

XLF

- XLF versions before 3.2.5.2 and 4.1.0.1 do not allow the target attribute on assumed size arrays in both fwrap.fpp and interf.fpp. If you get a compile time error message related to this, then add "#define OLDXLF" to the end of the fpp file in f90gl/fppincs. Note that if you do this then arrays for which C keeps a pointer and uses the array later may not work properly (for example, the glutdino example will not draw the sides of the dino).

Digital Fortran on Digital Unix

- with Digital Fortran 90 V4.1 you need to apply a patch to the compiler. Without it you will get an error message about no matching specific routine for the generic subroutine gluTessCallback when you compile the glutdino example. You can either :

- Download ECO 02 for DIGITAL Fortran for DIGITAL UNIX from <http://www.digital.com/fortran/downloads.html> and patch the compiler before building f90gl, or
- add `#define DIGITAL_NO_ECO_02` to the fpp file, but this is a hack that changes the API for `gluTessCallback` (see USRGUIDE). I only recommend this for someone who does not have root privileges (so can't patch the compiler) and can't get the sysadmin to patch it right away. You should reinstall f90gl without the hack once you finally do get the sysadmin to patch the compiler.

PACIFIC SIERRA

- currently does not work with the Pacific Sierra compiler on Linux. Check the f90gl web page for the latest information.

gfortran

- requires a version of gfortran newer than 10/23/2005

A.2.2 mf_key

Voici le fichier mf_key fournis par les développeurs de la librairie f90l, il explique comment trouver le code de votre système

This file describes the naming convention for the makefiles (mf*) and include files for the preprocessor (fppincs/fpp*).

In some cases a common makefile handles many systems; this is called mfmost. In fppincs, there is a fppmesa that is used by several of the systems with Mesa.

Otherwise, the naming convention for makefiles is mf12345 and preprocessor files is fpp12345 where

1. computer 8 x86 or Pentium
 - a alpha
 - g sgi
 - h hp
 - i Itanium (in some cases, 8 may apply to Itanium)
 - o sgi origin
 - p Power Macintosh
 - r rs6000
 - s sun
2. OS 4 SunOS 4
 - 5 Windows 95
 - a AIX
 - d Tru64 and other descendents of Digital Unix
 - h HPUX
 - i Irix
 - l Linux
 - n Windows NT
 - s Solaris
 - v OpenVMS
 - x MacOS X
3. f90 compiler a NASoftware
 - b Absoft
 - c Pacific Sierra
 - d Compaq (formerly Digital) Fortran
 - e EPC
 - f F
 - g SGI
 - h HP
 - i Intel

- j Fujitsu (including Lahey/Fujitsu on Windows)
 - l Lahey
 - m Microsoft
 - n NAGWare
 - o Salford
 - p PGI
 - s SunSoft
 - t PathScale
 - u Gnu g95 and gfortran (add 2 for gfortran)
 - v CVF (formerly DVF)
 - x xlf90
4. Opengl m Mesa
- n NVidia
 - o Native OpenGL
 - t MetroLink OpenGL for Linux
 - s SGI OpenGL for Windows
5. misc special designations a Linux a.out binaries
- 1 SunSoft versions before 2.0
 - 2 SunSoft version 2.0 or later
 - 2 Gnu gfortran, as opposed to g95
 - 3 SGI -n32
 - 6 SGI -64
 - 6 Intel ifc versions before 7.0
 - 7 Intel ifc version 7.0 or later
 - b Borland C compiler on Windows

Annexe B

fichier piece.f90

```
module callbacksPiece
  use varAffichage
  use plast2

  IMPLICIT DOUBLE PRECISION (A-H,O-Z)

  Integer :: imenuAppelantPiece

contains

  subroutine mainDisplayPiece()
    CALL DESPIECE
  end subroutine mainDisplayPiece

  subroutine comDisplayPiece()
    use plast2

    character :: TEXT*80,TEXT6*6
    integer :: NOZODI

    !permet d'effacer la fenêtre ,
    !il faut appelé cette méthode
    !à chaque fois si on ne veut pas
    !avoir un résultat très étrange
    call efface('com')
    !dessine les contour de la
    !sous fenêtre en bleu
```

```

    call desContourFenetre('com')

    NOZODI=2
    WRITE(TEXT6,'(I6)') NUMPI
    TEXT= 'DESSIN du BODY num'//TEXT6
    !écrit le TEXT en blanc dans la ligne
    !NOZODI de la fenêtre de commentaire
    CALL ECTX(NOZODI,TEXT)
end subroutine comDisplayPiece

subroutine comDisplayInfoPoint()
    character :: TEXT*80
    integer :: NOZODI

    call efface('com')
    call desContourFenetre('com')
    NOZODI = 2
    TEXT = 'cliquez sur un point du maillage&
           & pour avoir ses coordonnées '
    CALL ECTX(NOZODI,TEXT)
end subroutine comDisplayInfoPoint

subroutine sourisInfoPoint(button,state,x,y)
    !Coordonnées fenêtre (WXMIN...WYMAX)
    use varAffichage
    !etat et bouton
    use opengl_glut
    !subroutine de dessin
    use opengl_gl
    !certaines constante
    use init
    !utilisation des couleurs
    use couleur

    integer(gl cint), intent(inout) :: button, state, x, y
    real(kind=gldouble) :: xreel,yreel,xcr,ycr,xcw,ycw

    !on ne traite que l'enfoncement du bouton gauche
    if(state == GLUT_DOWN .AND. &
       & button == GLUT_LEFT_BUTTON ) then

```

```

!centre de la fenêtre
!(en tenant compte des sous fenêtres)
!ces coordonnées sont réelles,
!dans le repère écran.
XCW=(-0.99+0.50)/2.
YCW=(0.99-0.50)/2.

!centre de la figure en coordonnée réelles,
!dans le repère de base
XCR=(WXMIN+WXMAX)/2.
YCR=(WYMIN+WYMAX)/2.

!x et y sont en coordonnées relatives
!(par rapport au coin supérieur gauche
!de la fenêtre principale)
!conversion des coordonnées en coordonnées réelles
!dans le repère écran
xreel = real(x)/(WIDTH/2.)-1
yreel = real(y)/(HEIGHT/2.)-1

!transformation inverse des points pour avoir les
!coordonnées dans le repère de base:
!homotécie inverse + translation inverse
xreel = xreel*(rapport/0.80)-(XCW*RAPPORT-XCR)
yreel = -yreel*(rapport/0.80)-(YCW*RAPPORT-YCR)

IF (XREEL<WXMIN.OR.XREEL>WXMAX &
    & .OR.YREEL<WYMIN.OR.YREEL>WYMAX) THEN
    write(6,*)' clic hors cadre '
else
    write(6,*)' coordonnées du point : '
    write(6,*)' X = ',xreel, ' , Y = ', yreel

    !les points sont dessinés sur 3 pixels
    call glPointSize(TROIS)
    !les points sont dessinés en rouge
    call choixCouleur(KV2)
    !les points sont dessinés
    !dans la fenêtre principale
    call glutSetWindow(imainWin)

```

```

        ! dessin du point
        call glBegin (GL_POINTS)
        call glVertex3d(xreel ,yreel ,0.0 _gldouble)
        call glEnd()

        !on notera que ce point n'étant pas dessiner
        !dans une subroutine de callback ne sera pas
        !réaffiché avec la fenêtre
    end IF
end if
end subroutine sourisInfoPoint

subroutine menuPiece(iChoix)
    !contient bzoom, les identifiant
    use varAffichage
    !des fenêtres
    use opengl_glut
    !pour les callbacks de zoom
    use callbacksGeneral
    !pour les callbacks de l'appelant
    use callbacksAppelant

    integer , intent(in) :: iChoix

    !—————
    !  CONTOUR
    !—————
    if (ichoix==1) then
        !calcul du contour
        call contour()
        !dessin du contour
        call descont()

        !—————
        !  DEF POINT
        !—————
    elseif(iChoix==2) then
        call glutSetWindow(imainWin)
        call glutmousefunc(sourisInfoPoint)

        call glutSetWindow(icomWin)

```

```

    call glutdisplayfunc(comDisplayInfoPoint)
    call glutPostRedisplay()

    !—————
    !  ZOOM
    !—————
elseif(iChoix==3)then
    !on passe dans le mode zoom
    bzoom = .TRUE. !variable du module varAffichage
    call sauvZoom()
    !on affiche l'aide au zoom
    !dans la fenetre de commentaire
    call glutSetWindow(icomWin)
    call glutdisplayfunc(comDisplayZoom)
    call glutPostRedisplay()

    call glutSetWindow(imainWin)
    call glutmousefunc(zoomSouris)

    !—————
    !  ZOOM ARRIERE
    !—————
elseif(iChoix==4)then
    call restaurZoom()

    !on recadre
    call cadre()

    !on rafraichit la fenetre d'affichage
    call glutSetWindow(imainWin)
    call glutPostRedisplay()

    !—————
    !  FIN ZOOM
    !—————
elseif(iChoix==5)then
    !on sort du mode zoom
    bzoom = .FALSE.
    call remetInitZoom()

    !on change l'affichage des commentaires

```

```

call glutSetWindow (icomWin)
call glutdisplayfunc (comDisplayPiece)
call glutPostRedisplay ()

!on recadre
call cadre ()

!on rafraichi la fenetre d'affichage
call glutSetWindow (imainWin)
call glutPostRedisplay ()

!_____
!  FIN
!_____
elseif (iChoix==6) then

!fenetre de commentaire
call glutSetWindow (icomWin)
!attachement du callback
!d'affichage de l'appelant
call glutdisplayfunc (comDisplayAppelant)
!on raffiche la fenetre
!pour afficher le bon commentaire
call glutPostRedisplay ()

!fenetre principale
call glutSetWindow (imainWin)
call glutdisplayfunc (mainDisplayAppelant)
call glutPostRedisplay ()

!on se met sur la fenetre de menu
call glutSetWindow (imenuWin)
!on ne peut supprimer le menu courant
!que s'il existe (evite les crashes ...)
if ( glutGetMenu() /= 0 ) then
    !le menu etait attaché
    !au bouton droit de la souris ,
    !avant de la supprimer ,
    !il faut le détacher de ce bouton
    call glutDetachMenu (GLUT_RIGHT_BUTTON)
    !destruction du menu

```

```

        call glutDestroyMenu(glutGetMenu())
        !le menu de l'appelant n'a pas été détruit,
        !donc il suffit de le fixer à la fenêtre
        call glutSetMenu(imenuAppelantPiece)
        !on attache le menu courant,
        !qui est maintenant celui de l'appelant
        !sur le bouton droit de la souris
        call glutAttachMenu(GLUT_RIGHT_BUTTON)
    end if
endif
end subroutine menuPiece

end module callbacksPiece

SUBROUTINE PIECE
    use opengl_glut
    !le module callbacksPiece contient
    !les callbacks d'affichage
    !la variable permettant de sauver
    !le menu de l'appelant
    use callbacksPiece
    !contient les identifiant des fenêtre
    use varAffichage
    use plast2

    IMPLICIT DOUBLE PRECISION (A-H,O-Z)

    Integer , DIMENSION(1:20) :: IA
    Character *256 :: STR
    Integer :: NBCHOIX

    NBCHOIX=6
    STR='CONTOURDEF POINTZOOMZOOM ARRIEREFIN ZOOMFIN'
    IA(1)=7
    IA(2)=9
    IA(3)=4
    IA(4)=12
    IA(5)=8
    IA(6)=3

```

```

!on affiche la pièce numéro
!dans notre exemple
NUMPI=1
!(NUMPI se trouve dans le module plast2)

!pour pouvoir afficher la pièce
!on a besoin de cadrer
!pour cela on calcul les points
!bas, gauche (WXMIN,WYMIN)
!et haut, droit (WXMAX,WYMAX)
!de la pièce
call rechEspaceMin()
!puis on cadre avec les points précédents
call cadre()

!-----initialisation
!sauvegarde de l'identifiant du menu
call glutSetWindow(imenuWin)
imenuAppelantPiece = glutGetMenu()

!création du menu de piece
call menu(nbChoix,menuPiece,IA,STR)

!modification des callbacks d'affichage
!pour la fenêtre principale
call glutSetWindow(icomWin)
call glutdisplayfunc(comDisplayPiece)
!on raffiche la fenêtre une fois qu'elle est initialisée
call glutPostRedisplay()

!pour la fenêtre de commentaire
call glutSetWindow(imainWin)
call glutdisplayfunc(mainDisplayPiece)
call glutPostRedisplay()
!-----Fin initialisation

```

END SUBROUTINE PIECE

Annexe C

fichier despiece.f90

Annexe D

corps des méthodes de zoom

D.1 zoom

```
subroutine sauvZoom
  use callbacksPiece
  use varAffichage
```

```
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
```

```
REAL(KIND=GLFLOAT), DIMENSION(:,:) , ALLOCATABLE :: svgCoordCadres
integer :: iTaille , iok
```

```
!il faut sauvegarder le cadre courant
!si on a pas encore zoomé, il faut initialiser
if(allocated(coordCadres)) then
  !on avait déjà zoomé
  !on ajoute un zoom, on a besoin
  !d'un élément de plus dans le tableau
  iTaille = SIZE(coordCadres,DIM=1)+1
  !on désaloue svgCoordCadres pour être sur
  !de ne pas avoir de problème
  !d'allocation dynamique
  if (allocated(svgCoordCadres)) then
    deallocate(svgCoordCadres)
  end if
  !on aloue svgCoordCadres avec la nouvelle taille
  allocate(svgCoordCadres(iTaille,4),STAT=IOK)
  if(IOK == 0) then
```

```

        !on a put alouer la mémoire, on copie le tableau coordCadres
        svgCoordCadres = coordCadres
        !on désaloue coordCadres
        deallocate(coordCadres)
        !on alloue coordCadres avec la nouvelle taille
        allocate(coordCadres(iTaille,4),STAT=IOK)
        !on recopie le tableau svgCoordCadres
        !dans le tableau coordCadres
        coordCadres = svgCoordCadres
        !enfin on désaloue svgCoordCadres qui est maintenant inutile
        deallocate(svgCoordCadres)
    end if
else
    !c'est le premier zoom
    iTaille=1
    allocate(coordCadres(1,4),STAT=IOK)
end if
if(IOK == 0) then
    !on a put augmenter la taille du
    !tableau de sauvegarde des coordonnées
    !on remplit la dernière ligne du
    !tableau avec les coordonnées actuelles
    !fenêtre
    coordCadres(iTaille,1)=WXMIN
    coordCadres(iTaille,2)=WYMIN
    coordCadres(iTaille,3)=WXMAX
    coordCadres(iTaille,4)=WYMAX
    !write(6,*)'taille : ',iTaille,' sauvé : ' &
    !&,coordCadres(iTaille,1), coordCadres(iTaille,2),&
    !&coordCadres(iTaille,3), coordCadres(iTaille,4)
end if
if(IOK /=0) then
    !plusieur cas d'erreur possible => nouveau if
    !un erreur c'est produite lors de l'allocation, on arrête pas
    !le programme, on ne stock pas le zoom précédent
    write(6,*)'impossible de sauvegarder le zoom'
end if

end subroutine sauvZoom

```

```
subroutine restaurZoom
```

```
  use callbacksPiece
```

```
  use varAffichage
```

```
  !ce module inclut OPENGL.SDD
```

```
IMPLICIT DOUBLE PRECISION (A–H,O–Z)
```

```
REAL(KIND=GLFLOAT), DIMENSION(:,:), ALLOCATABLE :: svgCoordCadres  
integer :: iTaille,iok
```

```
  !il faut restaurer le cadre courant
```

```
  iTaille = SIZE(coordCadres,DIM=1)
```

```
  !on vérifie que l'on est pas revenu au zoom initial
```

```
  ! write(6,*)'test itaille : ',iTaille
```

```
  if (iTaille > 1) then
```

```
    if (allocated(svgCoordCadres)) then
```

```
      deallocate(svgCoordCadres)
```

```
    end if
```

```
    !on crée un tableau avec un élément de moins
```

```
    allocate(svgCoordCadres(iTaille – 1,4),STAT=IOK)
```

```
    if(IOK == 0) then
```

```
      !on peut sauvegarder le zoom précédent
```

```
      svgCoordCadres = coordCadres
```

```
      !on enlève le zoom courant du tableau
```

```
      iTaille = iTaille – 1
```

```
      if (allocated(coordCadres)) then
```

```
        deallocate(coordCadres)
```

```
      end if
```

```
      !on recopie le tableau sauvegardé (sans le zoom courant)
```

```
      !dans le tableau de zoom
```

```
      allocate(coordCadres(iTaille,4),STAT=IOK)
```

```
      coordCadres = svgCoordCadres
```

```
      deallocate(svgCoordCadres)
```

```
      !on restaure les coordonnées de la fenêtre
```

```
      !avec le dernier élément du tableau
```

```
      WXMIN = coordCadres(iTaille,1)
```

```
      WYMIN = coordCadres(iTaille,2)
```

```
      WXMAX = coordCadres(iTaille,3)
```

```
      WYMAX = coordCadres(iTaille,4)
```

```
      !write(6,*)'taille : ',iTaille,' restauré : ' &
```

```
      & ,WXMIN, WXMAX, WYMIN,WYMAX
```

```

    end if
    if(IOK /=0) then
        !plusieur cas d'erreur possible => nouveau if
        !un erreur c'est produite lors de l'allocation, on arrête pas
        !le programme, on ne stock pas le zoom précédent
        write(6,*)'impossible de sauvegarder le zoom'
    end if
end if

end subroutine restaurZoom

subroutine remetInitZoom
    use callbacksPiece
    use varAffichage
    !ce module inclut OPENGL.SDD

    IMPLICIT DOUBLE PRECISION (A-H,O-Z)

    REAL(KIND=GLFLOAT), DIMENSION(:,:) , ALLOCATABLE :: svgCoordCadres
    integer :: iok

    !on remet le zoom initiale
    if (SIZE(coordCadres,DIM=1)>1) then
        WXMIN = coordCadres(1,1)
        WYMIN = coordCadres(1,2)
        WXMAX = coordCadres(1,3)
        WYMAX = coordCadres(1,4)
        !on supprime les zoom sauvegardé
        if (SIZE(coordCadres,DIM=1)>2) then
            if (allocated(svgCoordCadres)) then
                deallocate(svgCoordCadres)
            end if
            !on a plus qu'un seul zoom a sauvegarder
            allocate(svgCoordCadres(1,4),STAT=IOK)
            if (allocated(coordCadres)) then
                deallocate(coordCadres)
            end if
            allocate(coordCadres(1,4),STAT=IOK)
            coordCadres = svgCoordCadres
        end if
    end if
end subroutine

```

```

        deallocate(svgCoordCadres)
    end if
end if

end subroutine remetInitZoom

subroutine finZoom
    use callbacksPiece
    use varAffichage
    !ce module inclut OPENGL.SDD

    IMPLICIT DOUBLE PRECISION (A-H,O-Z)

    REAL(KIND=GLFLOAT), DIMENSION(:,:), ALLOCATABLE :: svgCoordCadres
    integer :: iok

    !on supprime les zoom sauvegardé
    if (SIZE(coordCadres,DIM=1)>2) then
        if (allocated(svgCoordCadres)) then
            deallocate(svgCoordCadres)
        end if
        allocate(svgCoordCadres(1,4),STAT=IOK)
        if (allocated(coordCadres)) then
            deallocate(coordCadres)
        end if
        allocate(coordCadres(1,4),STAT=IOK)
        coordCadres = svgCoordCadres
        deallocate(svgCoordCadres)
    end if

end subroutine finZoom

```

D.2 écriture